

Progressive Web Apps (PWA)

What is Progressive Web Apps (PWA)?



- Modern web APIs + traditional progressive enhancement strategy = create cross-platform web applications.
- Work everywhere: desktop, tablet, and phone.
- Same user experience advantages as native apps.

Takes advantage of both web and native app features

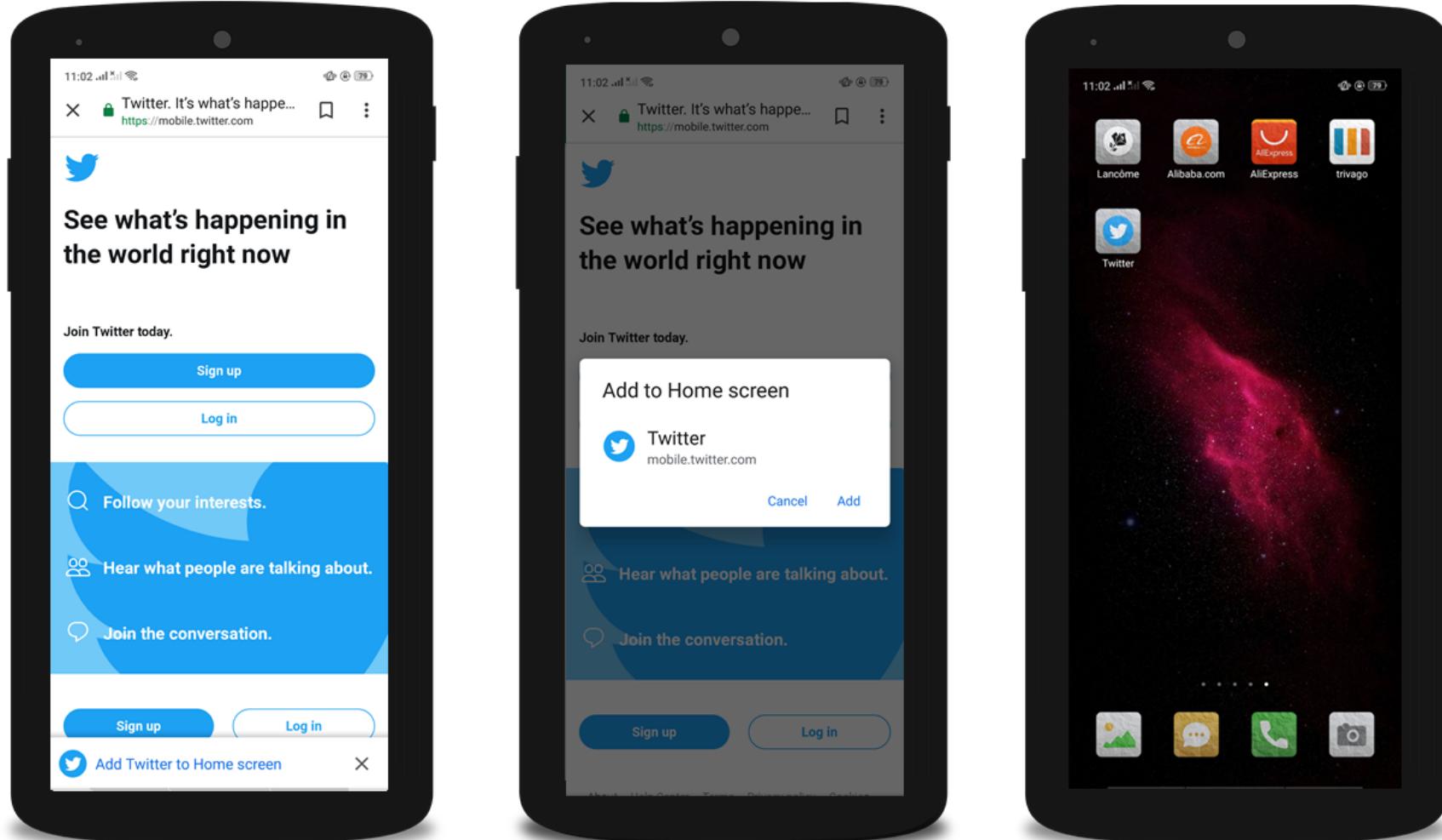
- As a web app
 - easier and faster to visit a website than install an application,
 - You can share web apps via a link.
- As a native app: can install it locally
 - Use *homescreen icons* to access the app
 - *Works offline*

Additional features

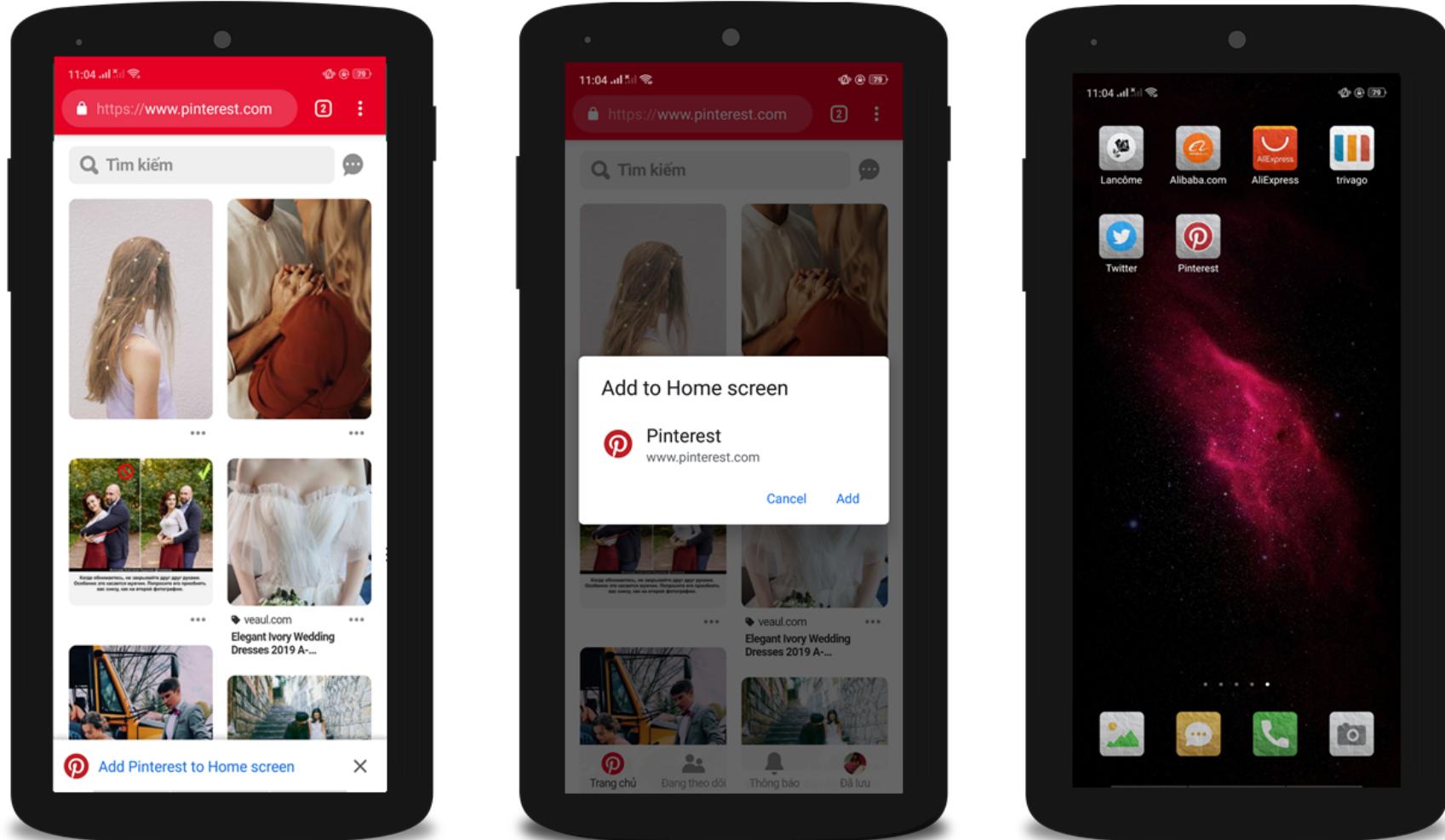
- *Discoverable*, can be found through search engines.
- *Progressive*, still usable on older browsers, and fully-functional on the latest ones.
- *Responsive*, usable on any device with a browser — phones, tablets, laptops, TVs, fridges, etc.
- *Safe*, connection between your device and app server is secured against any third parties trying to get access to your sensitive data.

Examples

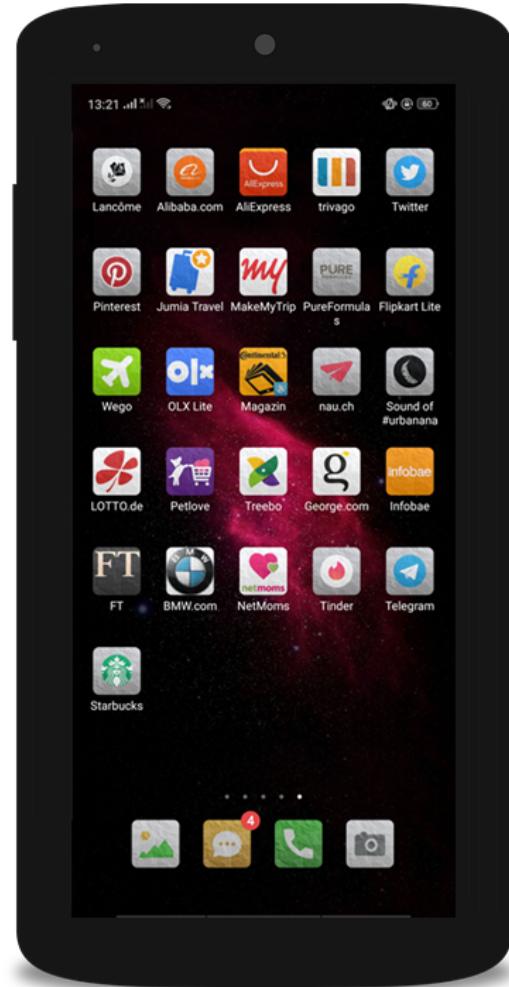
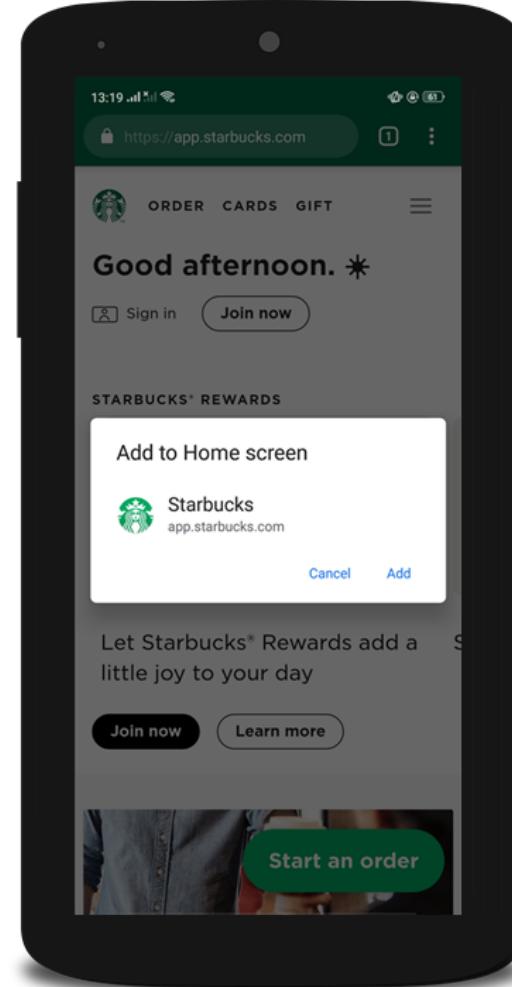
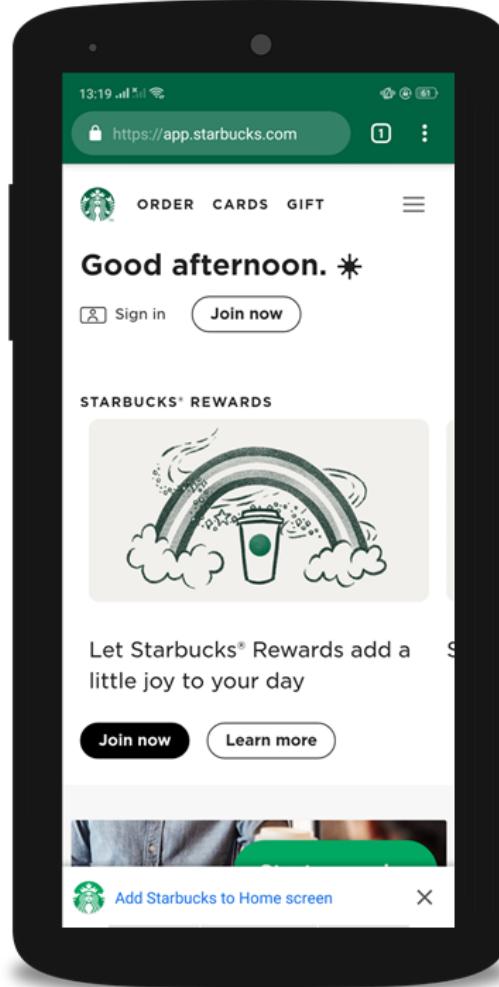
Twitter Lite



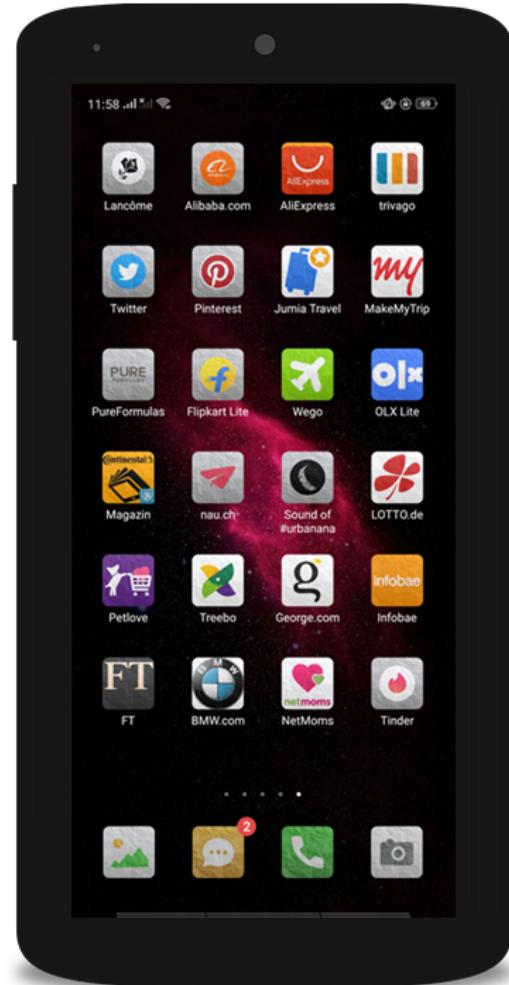
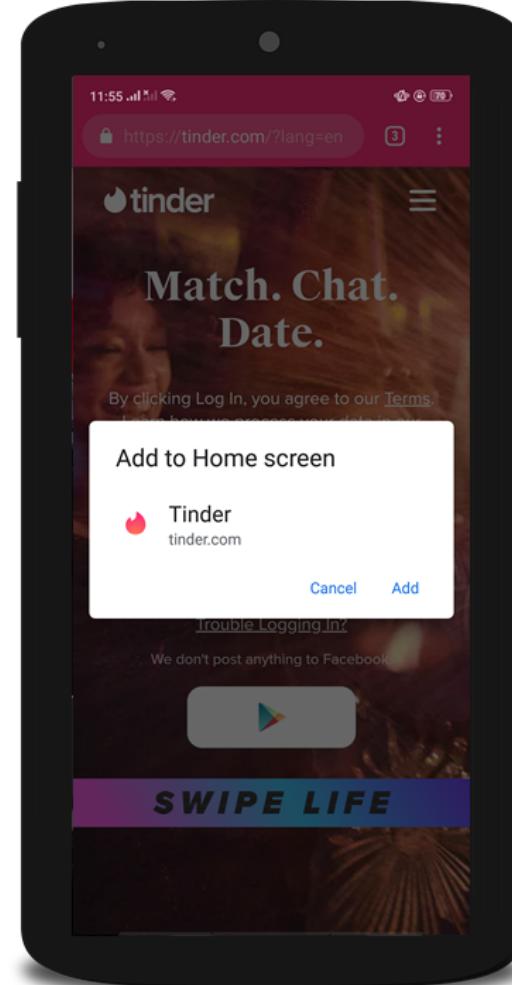
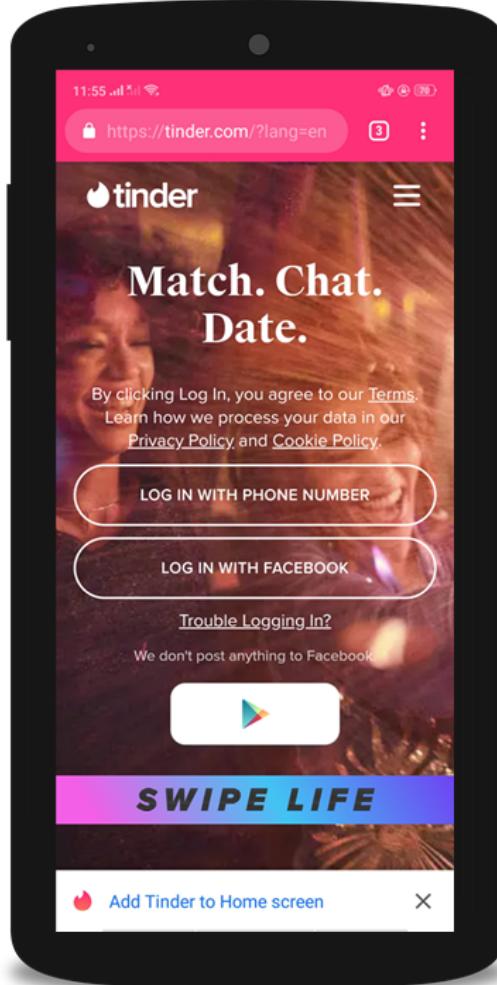
Pinterest



Starbucks



Tinder



Example: pet store

- We will continue with the 'pet store' example we had earlier.
- Turn it into a PWA
 - can install locally
 - add a shortcut to the home screen
 - work offline

The Pet Store - the <head>

```
<html>
<head>
  <title>Vue.js Pet Depot</title>
  <script src="https://unpkg.com/vue"></script>
  <script src="products.js"></script>
  ...
</head>
<body>
  ...
```

The product list - products.js

```
let products = [
  {
    "id": 1001,
    "title": "Cat Food, 25lb bag",
    "description": "A 25 pound bag of <em>irresistible</em>, ...",
    "price": 20.00,
    "image": "images/cat-food.png",
    "availableInventory": 10,
    "rating": 2
  },
  {
    "id": 1002,
    "title": "Yarn",
    "description": "Yarn your cat can play with for a very ...",
    "price": 2.99,
    "image": "images/yarn.jpg",
    "availableInventory": 7,
    "rating": 3
  },
  ...
]
```

The Pet Store - the Vue.js

```
var webstore = new Vue({  
  el: '#app',  
  data: {  
    products: products,  
    cart: [],  
    ...  
  },  
  methods: {  
    addToCart(product) {  
      this.cart.push(product.id);  
    },  
    ...  
  },  
  computed: {  
    sortedProducts() {...}  
    ...  
  },  
});
```

The Pet Store - the products

```
<div id="app">
  ...
  <div v-for="product in sortedProducts">
    <h2 v-text="product.title"></h2>
    <figure>
      
    </figure>
    <p v-html="product.description"></p>
    <p>Price: {{product.price}}</p>
    <p>Available stock: {{product.availableInventory}}</p>
    <button v-on:click='addToCart(product)'
            v-if='canAddToCart(product)'>
      Add to cart
    </button>
    ...
  </div>
  ...
</div>
```

Tidy up the formatting

```
.product {  
    border: 1px grey solid;  
    padding: 20px;  
    margin: 20px;  
    text-align: center;  
}
```

Yarn



Yarn your cat can play with for a very long time!

Price: 2.99

Available stock: 7

[Add to cart](#) [Buy now!](#)

★★★☆☆

Cat House



A place for your cat to play!

Price: 7.99

Available stock: 11

[Add to cart](#) [Buy now!](#)

★★★★★

Make PWAs installable

- A web manifest, with the correct fields filled in
- The website to be served from a secure (HTTPS) domain
 - We will use GitHub Pages, which provide HTTPS connection
- An icon to represent the app on the device
- A service worker registered, to make the app work offline

The manifest file

- Lists all the information about the PWA in a JSON format.
- It usually resides in the root folder of a web app.
- It contains useful information, such as
 - the app's name,
 - paths to different-sized icons (for example, as the home screen icon)
 - a background color to use in loading or splash screens.
- The manifest file has an extension of `.webmanifest` (`.json` also works).
- Link the manifest file in the `<head>` section of the html file:

```
<link rel="manifest" href="petstore.webmanifest">
```

petstore.webmanifest

```
{  
  "name": "My Pet Store PWA",  
  "short_name": "petstore",  
  "description": "my online pet store",  
  "icons": [  
    {  
      "src": "images/icon-512.png",  
      "sizes": "512x512",  
      "type": "image/png"  
    }  
  ],  
  "start_url": "index.html",  
  "display": "fullscreen",  
  "background_color": "#B12A34"  
}
```



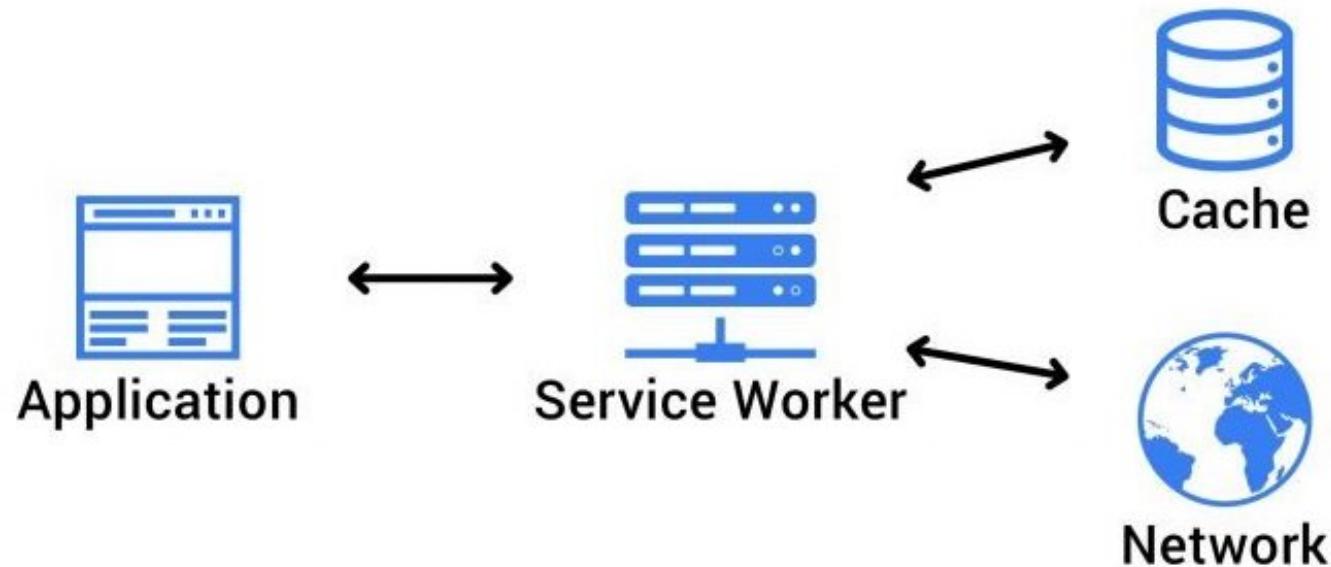
Open the app in browser

- It seems nothing has changed if you open the app in the browser now.
- If you go to the browser devtools >> 'Application' >> 'Manifest'
 - You can see Chrome recognised the `petstore.webmanifest` file
 - Showing values such as 'name', 'start URL', and 'background color'
- However, there is still an error 'Installability: No matching service worker detected'.

This screenshot shows the 'Application' tab in the Chrome DevTools. On the left, there's a sidebar with sections like 'Manifest' (which is highlighted with an orange circle), 'Service Workers', 'Clear storage', 'Storage' (with options for Local Storage, Session Storage, IndexedDB, Web SQL, and Cookies), 'Cache' (with Cache Storage and Application Cache), 'Background Services' (with Background Fetch, Background Sync, Notifications, Payment Handler, Periodic Background Scheduling, and Push Messaging), 'Frames' (with 'top'), and 'Icons' (with a checkbox for 'Show only the minimum safe area for maskable icons'). The main panel on the right displays the 'App Manifest' section, which lists the file `petstore.webmanifest`. It also shows the 'Installability' section with a warning message: 'No matching service worker detected. You may need to reload the page'. Other sections include 'Identity' (Name: My Pet Store PWA, Short name: petstore), 'Presentation' (Start URL: `index.html`, Theme color, Background color: #B12A34), 'Orientation' (Orientation: fullscreen), and 'Icons' (Primary icon, as used by Chrome, showing a stylized storefront icon).

Service workers

- Service Workers are a *virtual proxy* between the browser and the network.
- It can *cache* the files of a website and make them available when the device is offline.



- They run on a *separate thread* from the main JavaScript code
 - Don't have any access to the DOM structure.
- The API is *non-blocking*.
 - You can give a Service Worker something to work on, and receive the result when it is ready using a *Promise-based* approach.
- Service Workers can only be executed in secure contexts (meaning HTTPS)

Registering a Service Worker (in the main html file)

```
if('serviceWorker' in navigator) {  
    navigator.serviceWorker.register('service-worker.js');  
};
```

- If the service worker API is supported in the browser, it is registered using the `navigator.serviceWorker.register()` method.
- Its contents reside in the `service-worker.js` file, and can be executed after the registration is successful.
- When registration is complete, the `service-worker.js` file is automatically downloaded, then installed, and finally activated.

Caching the files (in `service-worker.js`)

- First, a variable for storing the cache name is created,
- Then, the files to be cached are listed in an array.

```
var cacheName = 'petstore-v1';
var cacheFiles = [
  'index.html',
  'product.js',
  'petstore.webmanifest',
  'images/yarn.jpg',
  'images/cat-litter.jpg',
  'images/laser-pointer.jpg',
  'images/cat-house.jpg',
  'images/icon-store-512.png'
  ...
];
```

Caching the files (in `service-worker.js`)

- Then we can cache the files in the 'install' event listener:

```
self.addEventListener('install', (e) => {
  console.log('[Service Worker] Install');
  e.waitUntil(
    caches.open(cacheName).then((cache) => {
      console.log('[Service Worker] Caching all the files');
      return cache.addAll(cacheFiles);
    })
  );
});
```

- The `self` here refers to the `window` object in JavaScript (the browser window).
- The service worker does not install until the code inside `waitUntil` is executed.
 - It returns a promise

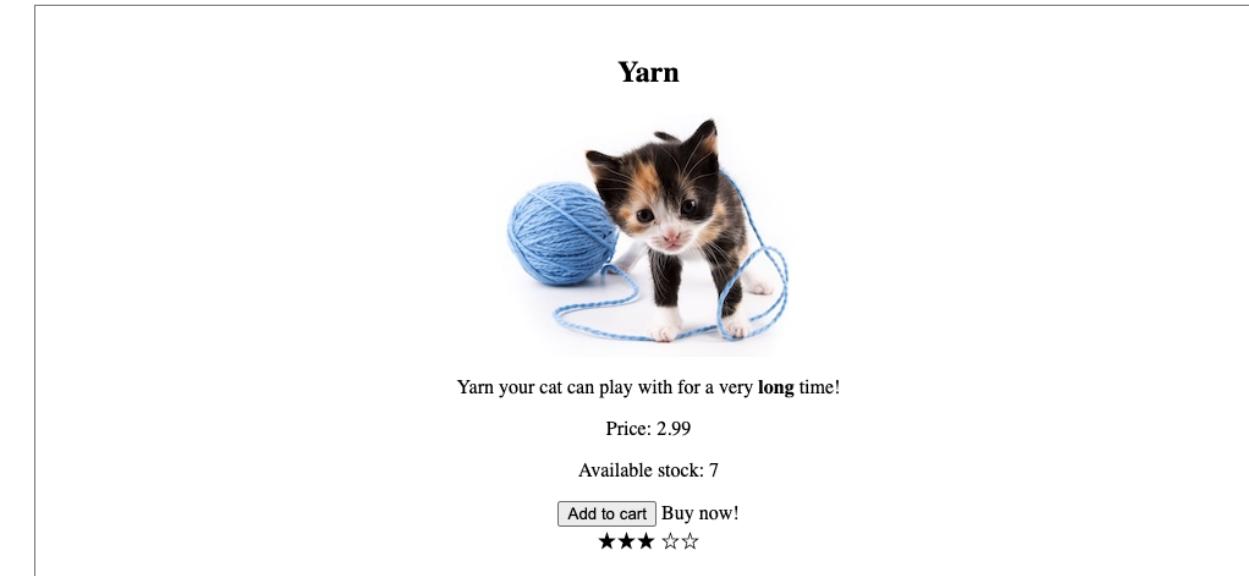
```
self.addEventListener('install', (e) => {
  console.log('[Service Worker] Install');
  e.waitUntil(
    caches.open(cacheName).then((cache) => {
      console.log('[Service Worker] Caching all the files');
      return cache.addAll(cacheFiles);
    })
  );
});
```

- The `caches` is a special 'Cache Storage' object for Service Worker to save data
 - Similar to 'local storage', but for caching only
- Here, we open a cache with a given name (`petstore-v1`),
- Then add all the files (listed in `cacheFiles`) to the cache,
 - So they are available next time it loads.

0 Checkout

Run the service worker

- If you open the app again in the browser, the page looks the same
- If you go to devtools >> Application >> Service Workers
- It should show that `service-worker.js` is activated and running
- If there is any error, make sure the file names in the list are correct.



The screenshot shows the Chrome DevTools Application tab. The 'Service Workers' section is highlighted with an orange box. It lists a single worker for the URL `http://127.0.0.1:5500/week16-pwa1/examples/`. The worker's source is listed as `service-worker.js`, it was received on `08/01/2021, 12:15:30`, and its status is `#1121 activated and is running`. Below this, there are sections for `Push`, `Sync`, and `Periodic Sync` messages, each with a corresponding input field and a 'Push' or 'Sync' button.

The cached files

- If you select 'Cache Storage' from the left pane, there should a cache named `petstore-v1`
- Clicking on `petstore-v1` will show the list of files that have been added to the cache in the top right pane
- Clicking on any of the file will show a preview in the bottom right pane

The screenshot shows the Chrome DevTools interface with the 'Application' tab selected. In the left sidebar, under 'Cache', the 'Cache Storage' section is expanded, showing a list containing 'petstore-v1 - http'. This list is highlighted with an orange oval. The main pane displays a table of cached files with columns for #, Name, Response Type, Content Type, Content Length, and Last Modified. The table includes rows for various files like 'index.html', 'products.js', and several image files. At the bottom right, there is a preview image of a cat sitting in a green pet bed.

#	Name	Respon...	Conte...	Conte...	Time ...
0	/week16-pwa1/examples/images/cat...	basic	image...	12,006	08/01...
1	/week16-pwa1/examples/images/cat...	basic	image...	25,138	08/01...
2	/week16-pwa1/examples/images/ico...	basic	image...	21,481	08/01...
3	/week16-pwa1/examples/images/las...	basic	image...	15,765	08/01...
4	/week16-pwa1/examples/images/yar...	basic	image...	20,770	08/01...
5	/week16-pwa1/examples/index.html	basic	text/h...	9,691	08/01...
6	/week16-pwa1/examples/products.js	basic	applic...	983	08/01...

Using the cached files

- Now that the files are cached, we can use the local files when starting the app rather than retrieving them from the server
 - This should be faster and also allow the app to work offline
- The ways to get this to work is to 'intercept' any `fetch` request
 - When the front end sends any `fetch` request to get data from the server
 - The service worker will redirect the request to the cache and return the file there
- This is achieved by listening to the `fetch` event:

```
self.addEventListener('fetch', (e) => {
  console.log('[Service Worker] Fetched resource '+e.request.url);
});
```

```
self.addEventListener('fetch', function (e) {
  e.respondWith(
    // check if the cache has the file
    caches.match(e.request).then(function (r) {
      console.log('[Service Worker] Fetching resource: ' +
        + e.request.url);
      // 'r' is the matching file if it exists in the cache
      return r
    })
  );
});
```

- Here, we respond to the fetch event with a function that tries to find the resource in the cache and return the response if it's there.
- The `FetchEvent.respondWith` method intercepts all `fetch` request
 - functions as a proxy server between the app and the network.

0  Checkout

Now the service worker will look all fine in the devtools

Yarn



Yarn your cat can play with for a very **long** time!

Elements Console Sources Network Application > | ⚙️ :

Application

- Manifest
- Service Workers**
- Clear storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Cache

- Cache Storage
- petstore-v1 - http://127.0.0.1:5500/week16-pwa1/examples/

Service Workers

Offline Update on reload Bypass for network

http://127.0.0.1:5500/week16-pwa1/examples/ [Update](#) [Unregister](#)

Source [service-worker.js](#)

Received 08/01/2021, 14:08:31

Status ● #1128 activated and is running [stop](#)

Push [Push](#)

Sync [Sync](#)

Periodic Sync [Periodic Sync](#)

However, there are still a few errors if you refresh the page

- When refreshed, all the files are loaded from the cache
 - Because we intercept every fetch request
- While html and images are there
 - Third-party files are missing
 - Such as the Vue.js and css for FontAwesome

The screenshot shows a product page with a cart item count of 1. The product details include price, available stock, and a message indicating it's out of stock. Below the product details is a star rating. The page title is "Checkout". The browser's developer tools are open, specifically the "Console" tab, which displays several error messages related to network requests for third-party resources like Unpkg and Cloudflare.

Product details:

- Price: {{product.price}}
- Available stock: {{product.availableInventory}}
- Add to cart (disabled)
- All out! Only {{product.availableInventory - cartCount(product.id)}} left! Buy now!
- ★☆

Checkout

First Name:

Last Name:

Address:

City:

Console tab (Developer Tools):

- [Service Worker] Fetching resource: https://unpkg.com/vue service-worker.js:27
- ⚠ The FetchEvent for "https://unpkg.com/vue" resulted in a network error response: an object that was not a Response was passed to respondWith().
- [Service Worker] Fetching resource: https://cdnjs.cloudflare.c service-worker.js:27 om/ajax/libs/font-awesome/5.15.1/css/all.min.css
- ⚠ The FetchEvent for "https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css" resulted in a network error response: an object that was not a Response was passed to respondWith().
- [Service Worker] Fetching resource: http://127.0.0.1:5500/week service-worker.js:27 16-pwa1/examples/products.js
- ✖ GET https://unpkg.com/vue net::ERR_FAILED index.html:5
- ✖ GET https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/al l.min.css net::ERR_FAILED index.html:7
- ✖ ▶ Uncaught ReferenceError: Vue is not defined at index.html:125 index.html:125

Caching new files

- While it is possible to add all the missing files to the caching file list
- The alternative is to download missing files dynamically and add them to cache

```
self.addEventListener('fetch', function (e) {
  e.respondWith(
    caches.match(e.request).then(function (r) {
      // Download the file if it is not in the cache,
      return r || fetch(e.request).then(function (response) {
        // add the new file to cache
        return caches.open(cacheName).then(function (cache) {
          cache.put(e.request, response.clone());
          return response;
        });
      });
    });
  );
});
```

- `fetch(e.request)` : if the file is not in the cache, we use another fetch request to download it,
- `caches.open(cacheName).then(...)` : then store the response in the cache so it will be available there next time it is requested.
- This allows us to respond to every single request with any response we want:
 - This includes other request type besides `GET`
 - The service worker have full control of the response (and can be potentially used for malicious purpose)

Now the missing files are added to the cache when the page reloaded

- The files for FontAwesome and Vue are now added to the cache

Vue.js Pet Depot

0 🛍 Checkout

Yarn

Elements Console Sources Network Application > | : X

#	Name	Respon...	Content...	Content...	Time ...
0	/week16-pwa1/examples/images/c...	basic	image...	12,006	08/01...
1	/week16-pwa1/examples/images/c...	basic	image...	25,138	08/01...
2	/week16-pwa1/examples/images/ic...	basic	image...	21,481	08/01...
3	/week16-pwa1/examples/images/la...	basic	image...	15,765	08/01...
4	/week16-pwa1/examples/images/y...	basic	image...	20,770	08/01...
5	/week16-pwa1/examples/index.html	basic	text/h...	9,691	08/01...
6	/week16-pwa1/examples/petstore....	basic	applic...	355	08/01...
7	/week16-pwa1/examples/products.js	basic	applic...	983	08/01...
8	/ajax/libs/font-awesome/5.15.1/css...	opaque	text/c...	10,491	07/01...
9	/ajax/libs/font-awesome/5.15.1/we...	cors	applic...	80,300	07/01...
10	/vue	opaque	applic...	0	07/01...

Application Storage Cache

Manifest Service Workers Clear storage

Local Storage Session Storage IndexedDB Web SQL Cookies

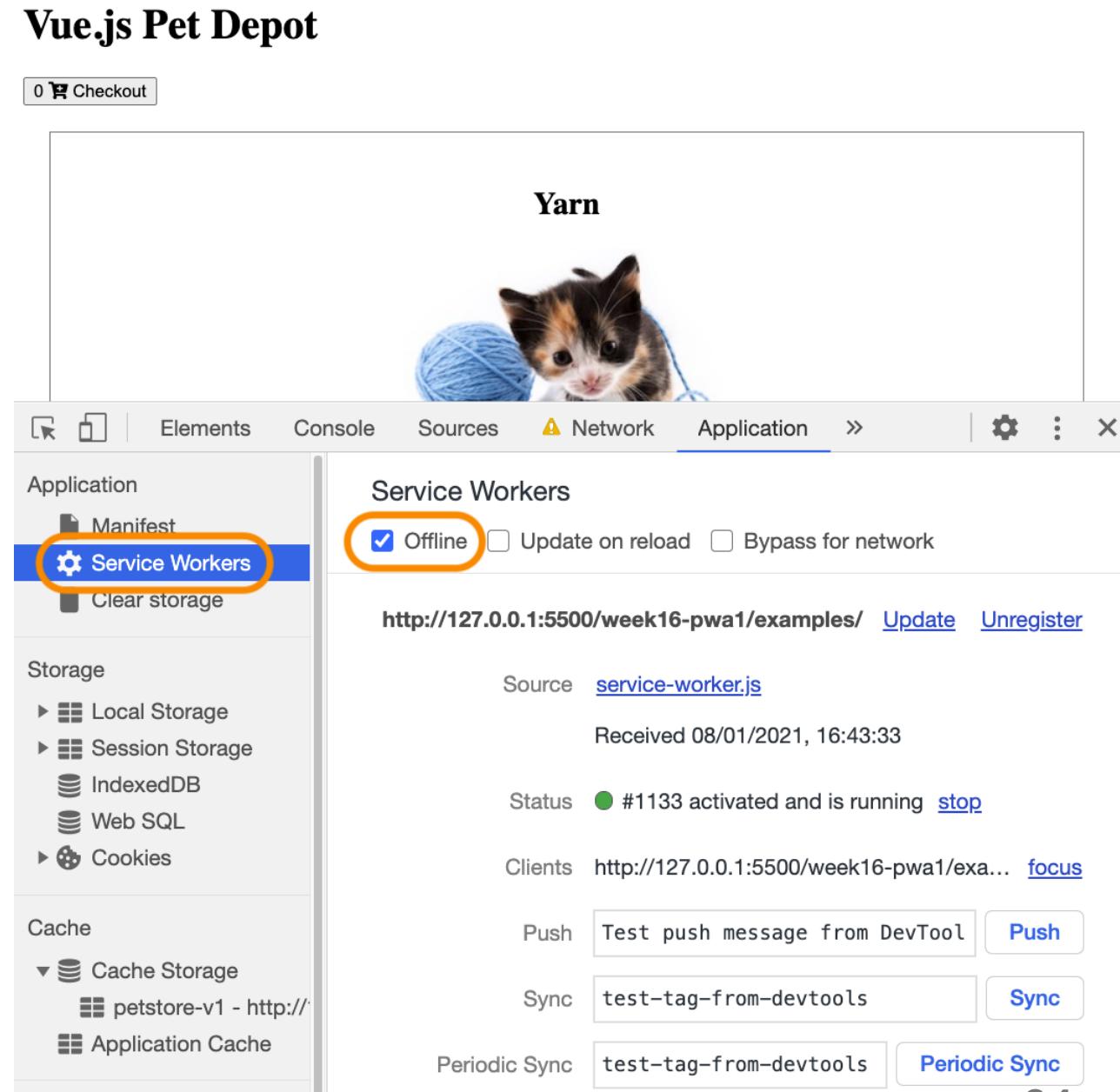
Cache Storage

petstore-v1 - http://

Application Cache

You can also test this in the 'Service Workers' pane with the 'Offline' option

- This disconnects the network
- And the app should still work fine after refresh

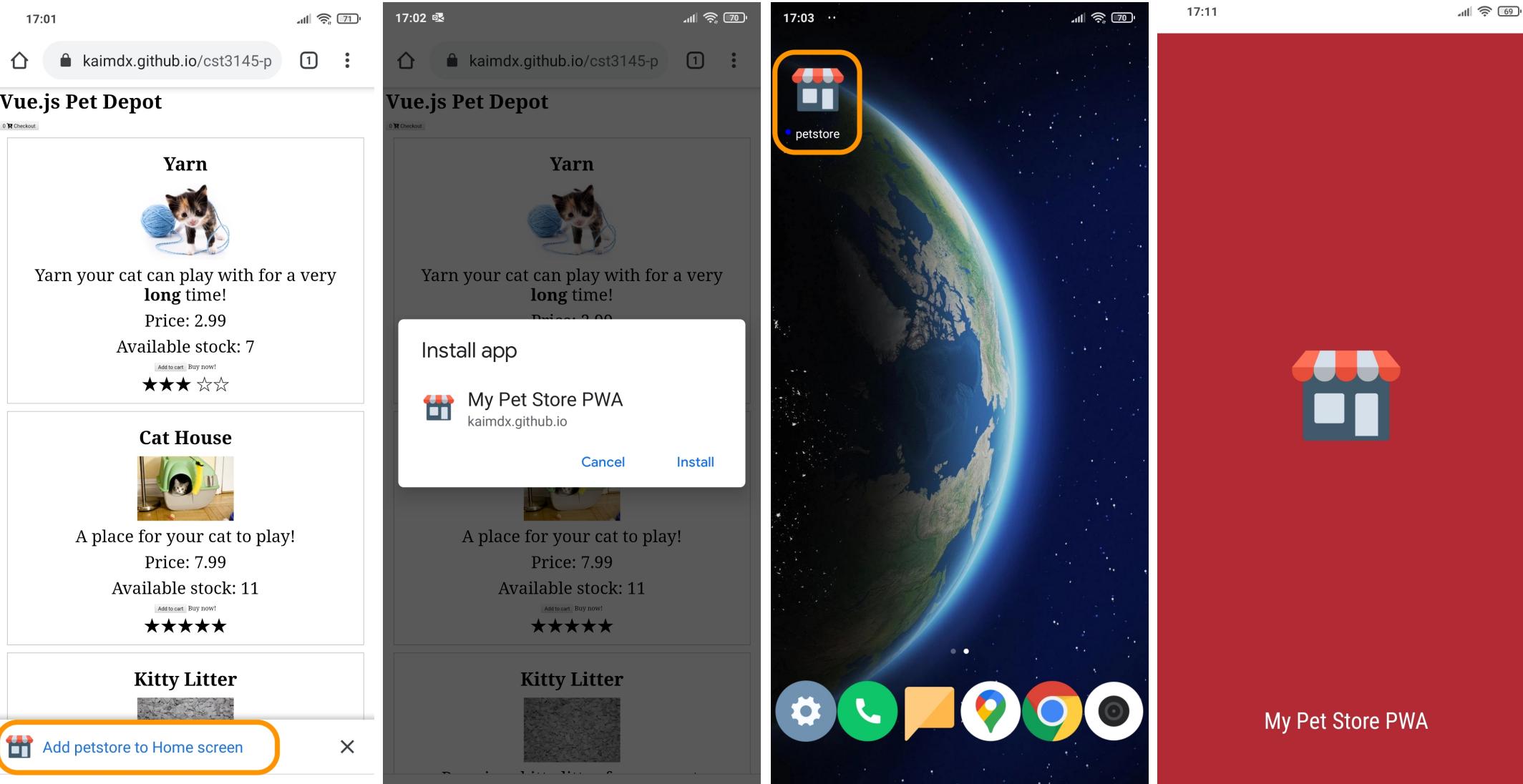


Now we meet all the requirements to make PWA installable

Just need to upload it to GitHub Pages

- A web manifest, with the correct fields filled in
- The website to be served from a secure (HTTPS) domain
 - We will use GitHub Pages, which provide HTTPS connection
- An icon to represent the app on the device
- A service worker registered, to make the app work offline

Now you can install it on your phone



Reading

MDN: Introduction to progressive web apps